UoA-TE-16-148

Making Big Data, Safe Data: A Test Optimization Approach

15 June 2016

Ricardo Valerdi, Associate Professor

Eddie Enhelder

University of Arizona

"This material is based upon work supported by the Naval Postgraduate School Acquisition Research Program under Grant No. N00244-14-1-0022. The views expressed in written materials or publications, and/or made by speakers, moderators, and presenters, do not necessarily reflect the official policies of the Naval Postgraduate School nor does mention of trade names, commercial practices, or organizations imply endorsement by the U.S. Government."

The research presented in this report was supported by the Acquisition Research Program of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

UoA-TE-16-148

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Making Big Data, Safe Data: A Test Optimization Approach

15 June 2016

Ricardo Valerdi, Associate Professor

Eddie Enhelder

University of Arizona

"This material is based upon work supported by the Naval Postgraduate School Acquisition Research Program under Grant No. N00244-14-1-0022. The views expressed in written materials or publications, and/or made by speakers, moderators, and presenters, do not necessarily reflect the official policies of the Naval Postgraduate School nor does mention of trade names, commercial practices, or organizations imply endorsement by the U.S. Government."

THIS PAGE LEFT INTENTIONALLY BLANK

Executive Summary

The following report outlines a procedure and algorithm to optimize the potential knowledge gained about a complex system when performing robustness testing and faced with a set of constraints. In particular, this project was catalyzed by the need to put a value on testing. Included with this project report is a proof of concept created in MS Excel utilizing its VBA developer tool. In short, a test network is created by establishing test relationships and then assigning each an expected knowledge value. With these values and an understanding about the relationships between the tests, an optimization about the total potential knowledge of the system can be acquired while minimizing testing costs and/or effort.

THIS PAGE LEFT INTENTIONALLY BLANK

Table of Contents

Introduction 1

The Problem of Limited Resources 1

Knowledge vs. Time, Funding, and Testing Capabilities 2

Relationships Between Tests 2

Project Motivation 3

Solution 4

Overview 5

Knowledge Gradients 5

Networks 6

Nodes 7

Arcs 7

Network Structures 8

Methods & Procedure 9

Test Optimization Process 9

Step 1: Establishing the Test Network 9

Step 2: Knowledge Gradient Index 10

Step 3: Assigning Expected Effort and Cost 10

Step 4: Knowledge Gradient Scalars 10

Step 5: Inputting the Network Data 11

Step 6: Run Algorithm 11

Step 7: Perform Suggested Tests 12

Step 8: Input Test Results 12

Step 9: Repeat Steps 6 – 8 12

Assumptions & Limitations 12

Proof of Concept Simplification 13

Programming Language 13

Proof of Concept 13

Step 1: Establishing the Test Network 13

Steps 2 & 3: Scoring the Knowledge Gradient Index (KG) and Assigning Expected Effort and Cost 14

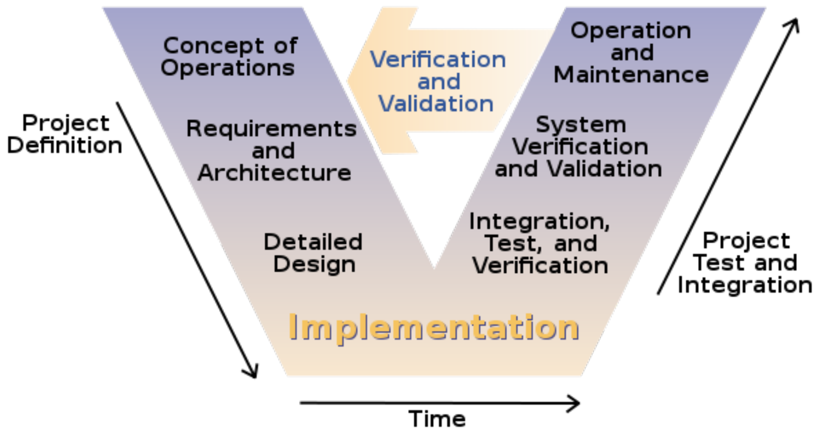
Step 4: Knowledge Gradient Scalars (KGS) 15

Step 5: Inputting the Network Data 16

Step 6: Run Algorithm 16
Step 7: Perform Suggested Tests 17
Step 8: Input Test Results 17
Step 9: Repeat Steps 6 – 8 18
Results 18
Conclusion 21
Appendices 28
Source Code – Run Algorithm Button 28
Source Code – Clear Button 32
References 33

Introduction

Test and evaluation (T&E) is a critical component in systems engineering. In many circumstances T&E is used to verify that a given system and its components are meeting the requirement specifications assigned to each and every level of the design. This is represented in the widely used Vee Model in systems engineering, see image below.


Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

[1]

This type of T&E is typically very specific; all tests to be performed are known in advance and all tests will usually be performed. In the project definition stage, or left side of the Vee model, these tests will be designed and the expected cost and effort associated with them will be determined. With this type of T&E situation there is very little variance in cost and effort, assuming all tests pass without problems. Essentially, you know exactly what tests will be performed, when they will be performed, how much they will cost, and what effort will be needed to complete them. Additionally, the knowledge gained from testing is known because all tests must pass, at least within some acceptable tolerances.

This is not the type of T&E situation that this project is attempting to optimize; we will optimize a situation that tests the robustness of a system.

> *"Robustness is defined as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions. [IEEE Std 24765:2010]"* [2]

This type of testing requires a selection strategy to choose the best tests to be performed because there are more tests possible than resources will permit.

The Problem of Limited Resources

In December of 2000 a study was concluded by the Defense Science Board Task Force on Test and Evaluation Capabilities. This study states, "*Testing must be robust and thorough otherwise a false sense of confidence and security can be generated when critical tests are waived or avoided. The pursuit of the reformed acquisition goals of "faster, better, cheaper" must not compromise thorough, robust, objective testing.*" [3] A common problem arises when testing the robustness of a system, having more tests that can be performed than available resources.

This begs the question:

Which tests should be performed to gain the most knowledge about a system when there is a limited amount of time, funding, and/or testing capabilities?

Knowledge vs. Time, Funding, and Testing Capabilities

With little convincing, one should understand that performing a test takes some amount of time, and every time you complete a test you gain some amount of knowledge or information about the system being tested. Furthermore, a limited amount of testing capabilities will also limit your ability to run multiple tests concurrently. For example, if you have 100 tests that you want to perform to analyze your system in the least amount of time possible (assuming there are enough funds to do so,) then you should perform all tests concurrently. This would make the total testing period as long as the longest test, assuming all test are passing. There is a clear limitation here with a testers testing capabilities (equipment and/or test subjects).

If you have enough testing capabilities to run 10 tests at a time, this means that you would have to run 10 rounds of testing to complete all 100 tests. Now your testing period has significantly increased.

What if you can't run all of the tests even if you had unlimited time? If you can only run at best 50 of the 100 possible tests due to a budget constraint, which tests should you perform? This leads us to yet another problem, the value of testing.

The Defense Science Board Task Force *"…found that the most significant capability missing in the T&E community is the ability to measure the "value of testing.""* [3] If we think about this problem intuitively, we could say the *"value of testing"* is the information or knowledge we receive from performing a given test. Thus the unit of measurement for *"value of testing*" is the knowledge gained. So if we know that the goal is to gain the most possible knowledge about the system, the obvious answer is to choose the tests which yield the most knowledge. But how do we determine this? To do so, we will introduce the concept of a knowledge gradient index later in this paper. For now, let's consider the problem of ranking and selection.

According to Powell and Ryzhov's *Optimal Learning (2012)* ranking and selection problems occur in many settings, and assessing and selecting for the best possible alternatives (tests) in such settings requires resources. These resources are assumed to be constrained by budget, whether this budget is in the currency of time or dollars. [4] Facing these testing limitations, the only solution to maximize the knowledge of a system under the constraints of time, money, and testing capabilities, is to optimize decision making regarding testing strategies. This project addresses how to optimize these testing strategies through the use of an algorithm.

Relationships Between Tests

Another element of difficulty to add to this problem is test relationships between tests. These relationships occur when there is a dependency between one test result and the knowledge to be gained from other correlated tests. According to Frazier (2009), "*We say that there is dependence between alternatives in [a Ranking & Selection] (R&S) problem* *if, when we measure one alternative, we learn something about the other."* [5] The results of one test has the potential to either increase or decrease the assumed knowledge gained from performing any other related tests.

Here is a simple example of test dependency:

> You are testing how bullet proof a car door is. You have selected 4 possible tests to run:
>
> 1) shoot the door with a .22 caliber round
> 2) shoot the door with a .38 caliber round
> 3) shoot the door with a .45 caliber round
> 4) shoot the door with a .50 caliber round
>
> The first test you choose to run is test 2, because expert opinion has determined you get the most knowledge for the cost to perform this test. The results of the test show that the car door withstood the bullet with no issue. Knowing this result, how important is it to perform test 1? Due to test dependency, the knowledge gained from the initial test, makes performing test 1 obsolete. If the door can withstand a .38 caliber bullet, then it will certainly withstand a .22 caliber bullet. We clearly see that performing test 1 will yield no increase in knowledge about the door.
>
> Similar logic can be applied to tests 3 and 4. Since this test dependency exists, we know the results of test 2 will change the knowledge gained from 3 and 4 in a positive upward direction. In conclusion, to gain the maximum knowledge about a system, we should always update the potential knowledge that can be gained of all related tests.

Project Motivation

DoD's reliance on network centricity has increased the need for secure cyberspace, which is difficult to test due to schedule slippages and evolving operational needs. To mitigate this, we propose an analytical method to optimize the test planning process, thereby accelerating test schedules while improving test coverage. [6] One constraint of interest not mentioned in this statement is cost, which will be a focal point of this project. Hence we focus on two research questions:

(1) How can testers maximize knowledge about the system under test efficiently?
(2) How should test strategies change to optimize available resources?

These two questions have been the motivation for the following research question this project is concerned with:

Which tests should be performed to gain the most knowledge about a system when there is a limited amount of time, funding, and/or T&E resources?

The solution provided in this paper will avoid limitations faced by Test Engineers when applying Design of Experiments (DOE) to deterministic systems. Applying DOE to deterministic systems – where the same inputs generate the same outputs over time – is often sufficient. However, when dealing with nondeterministic systems where emergent behaviors are the norm, DOE has significant limitations. Such limitations are based on the following:

(1) The assumption that the entire trade space is known;
(2) The ability to automatically re-plan the test strategy as emergent behaviors are observed;
(3) The assumption that the value of each test is constant regardless of the feature it is intended to test; and
(4) The assumption that the cost of each test (e.g., time, money, equipment) is the same." [6]

Limitations 2 and 3 should be clear when compared to the bulletproof car-door example mentioned previously.

Solution

We propose the following 6 steps to solve this problem:

Test Planning Step 1: Prioritize system and/or mission requirements for the system under test.

Test Planning Step 2: Define and quantify the cost, c, of running each test. This will be calculated using a parametric cost model that considers the complexities of net centric systems and the resources (in terms of people, equipment, and facilities) needed to execute each test.

Test Planning Step 3: Determine $\bar{\theta}_d^0$, the initial estimate of the expected reward for making decision d, where each decision involves selecting a specific test that should be executed. In this case, a reward can be considered to be the generation of new knowledge about the system under test.

Test Planning Step 4: Determine $\bar{\sigma}_d^0$, the initial estimate for the standard deviation of $\bar{\theta}_d^0$. The standard deviation is based on the fact that the expected rewards are normally distributed. Higher values of $\bar{\sigma}_d^0$ indicate lower confidence in the decision under consideration.

Test Planning Step 5: Execute the knowledge gradient algorithm to calculate the knowledge gradient index (KG) for feasible decisions. Since the KG jointly optimizes three criteria: value, cost, and knowledge acquired, it can be used to develop a prioritization of the tests to be performed. At this stage in the testing process, the first phase of testing is performed and data [is] collected about the performance of the system.

Step 6: Execute Bayesian updating algorithm to re-calculate KG index based on new information (e.g., test results, shifting evolving mission requirements, test costs, test facility availability, etc.) and provide an updated test strategy based on a recommended prioritization.

Given the existing research, these steps will not be addressed entirely in this project. In particular, steps 3 and 4, which partly tie into step 6, have been previously worked on by Amjad Chatila in his paper "An Introduction to Qualitative Resilience," where he discusses "Beliefs and decisions using probabilistic graphs to adapt Bayesian Belief Networks." [7]

To validate the functionality of the Algorithm created, we have developed a simplified solution. In order to execute these steps and provide a proof of concept, the following assumptions and simplifications have been made:

- Step 1: The testing requirements shall be prioritized by KG/$
- Step 2: Cost will be randomly assigned to a pool of tests rather than the development of a parametric model
- Step 3: The expected knowledge reward for performing a given test will be randomly generated from a predetermined KG index
- Step 4: For simplification of performing step 6, standard deviation of the KG index won't be considered
- Step 5: Only knowledge will be optimized for KG index working within the budget and resource constraints, for the purpose of this project value has been defined as knowledge
- Step 6: Bayesian probability will be disregarded from the updating as a byproduct of the assumption in step 4, expected values from step 3 will instead be assigned as constants

This proposed solution wouldn't be necessary if the number of tests were small. If this was the case, it could easily be solved by hand in little time. However, when performing hundreds of tests, this is an impossible task without the help of an algorithm.

The foundation of this algorithm solution is to take a pool of tests and map their relationships into a network structure. Using this network map, we can create a logical 9 step process coupled with a network algorithm to optimize the knowledge gained of a system that is undergoing robustness testing. This 9 step process expands on Valerdi's 6 step solution. In the following section we will review network structures.

Overview

Knowledge Gradients

A brief discussion about knowledge gradients (KG) is in order, however, the development of one and its policies as it applies to robustness testing is beyond the scope of this project. We only aim to create an algorithm that can optimize a KG index under a set of testing constraints.

The definition of KG is well stated in Powel (2012):

"The knowledge gradient is the name we apply to the simple idea of measuring the alternative that produces the greatest value, in expectation, from a single observation," and *"the concept of the knowledge gradients [can be applied to] ranking and selection problems."*

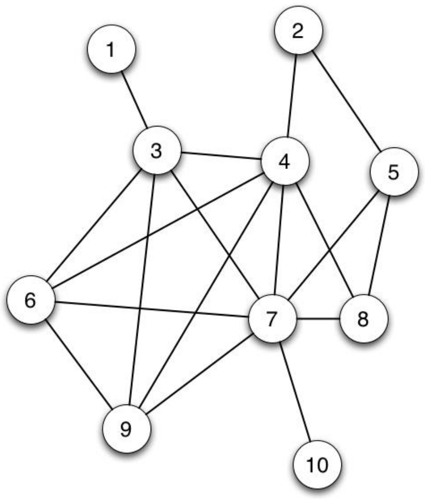
To develop a KG index, one should start by defining a KG policy, in this case, a "correlated KG policy." Frazier explains this in his dissertation:

> *"…we develop a general class of practical and theoretically well-founded information collection policies known as knowledge-gradient (KG) policies. KG policies have several attractive qualities: they are myopically optimal in general; they are asymptotically optimal in a broad class of problems; they are flexible and may be computed easily in a broad class of problems; and they perform well numerically in several well-studied ranking and selection problems compared with other state-of-the-art policies designed specifically for these problems."* [5]

Robustness testing results vary from system to system, requiring KG to be uniquely created for each. For this reason, this project only provides a generic KG index to validate the algorithms functionality.

Networks

The simplest definition of a network is "a group or system of interconnected people or things," [8] in this case, tests. The image below is a visual representation of a network. The numbered circles, or nodes, represent a given test. The lines that connect the nodes are the arcs of the network.



[9]

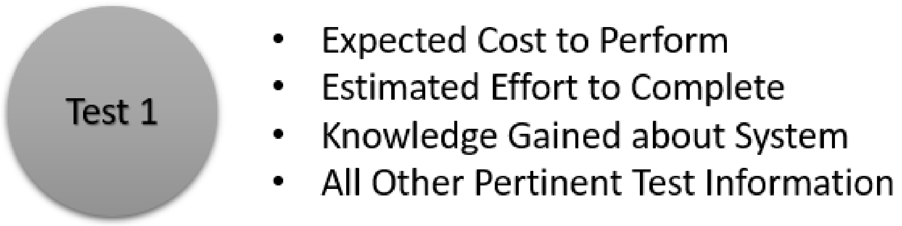
Computers and programming languages can't understand a network in this format. Instead, we represent the network in the form of an adjacency matrix, as seen below.

Test #	1	2	3	4	5	6	7	8	9	10
1	0	0	1	0	0	0	0	0	0	0
2	0	0	0	1	1	0	0	0	0	0
3	1	0	0	1	0	1	1	0	1	0
4	0	1	1	0	0	1	1	1	1	0
5	0	1	0	0	0	0	1	1	0	0
6	0	0	1	1	0	0	1	0	1	0
7	0	0	1	1	1	1	0	1	1	1
8	0	0	0	1	1	0	1	0	0	0
9	0	0	1	1	0	1	1	0	0	0
10	0	0	0	0	0	0	1	0	0	0

This adjacency matrix is representative of the visual network above; it is left to the reader to make the connection between the two. Note that 1 represents an arc (line between nodes) and none otherwise.

Nodes

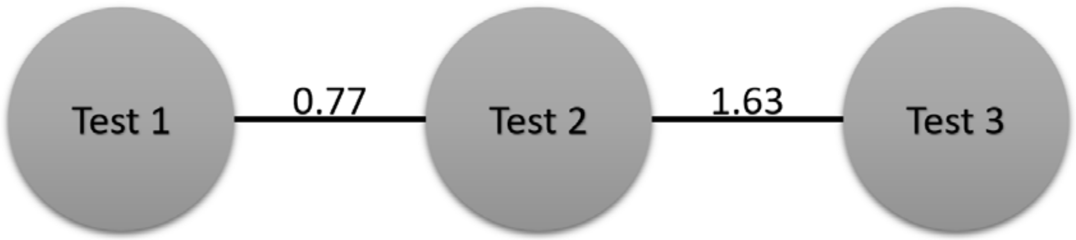
As just stated, the nodes represent a specific test. Each node should contain characteristics about the test, i.e. the identification number, the expected cost, the estimated effort, and the knowledge the test will provide about the system. I will briefly mention here that a well-defined knowledge gradient (KG) policy needs to be created to represent the knowledge gained by performing a given test.



- Expected Cost to Perform
- Estimated Effort to Complete
- Knowledge Gained about System
- All Other Pertinent Test Information

Arcs

The arcs or connections between the nodes not only represent a relationship between two or more nodes, but also contain a value that changes a characteristic of a related node. In our case, these are called knowledge gradient scalers (KGS). Using the 3 node network below, you can see that if you performed Test 2 with X results, that test would scale a characteristic (KGS) of the neighboring nodes/tests. For the problem we are solving, we are changing the value of knowledge that could be gained from the neighboring tests based on our current test result. If each of the 3 tests had a KG index value of 10, and we obtained the X results from Test 2, Test 1 would have a new knowledge value of 7.7, and Test 3 would be 16.3. Thinking back to the bulletproof car door example, and seeing this mathematical result, it is easy to see how this works.



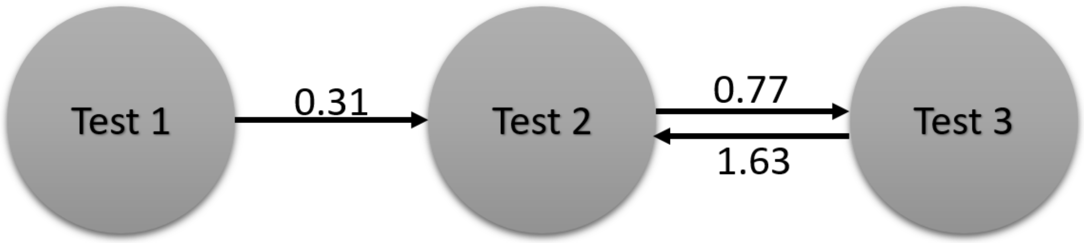
We have the same issue here as we did with the network representation; computers and programming languages can't understand this arc-node relationship in this format. Once again, we can represent the network in the form of an adjacency matrix. We leave it to the reader to make the connection between the two.

Test #	**1**	**2**	**3**
1	0	0.77	0
2	0.77	0	1.63
3	0	1.63	0

Network Structures

The next thing to understand about networks and the node-arc relationship is that up until now we have only shown examples of undirected networks. This means that the KGS values are a two-way street. Looking at the previous example, the 3 test network, if you had performed Test 1 first, it would have scaled Test 2 to a value of 7.7. For the problem we are discussing, it would be quite rare if two different neighboring tests had an equal scaling effect on each other's KG index value.

In most cases we will be dealing with directed networks, meaning a one-way street, and possibly a one-way street in two directions. Look at the following example:



You see that the arcs now have arrows suggesting a direction, hence a directed network. If test 1 is performed first, the result will decrease the KG index value from performing

Test 2. If you perform Test 2 first, the results have no effect on the KG index value from Test 1. It should be apparent how the two one-way streets (arcs) between Tests 2 and 3 affect the knowledge to be gained from either test, depending on which test is performed first.

There is some basic terminology to know about node relationships. Looking at Test 2, the connecting nodes that point *to* it, are called the predecessors. The connecting nodes that have arrows pointing *away* from Test 2 are called successor nodes. We will use these terms for the remainder of this paper.

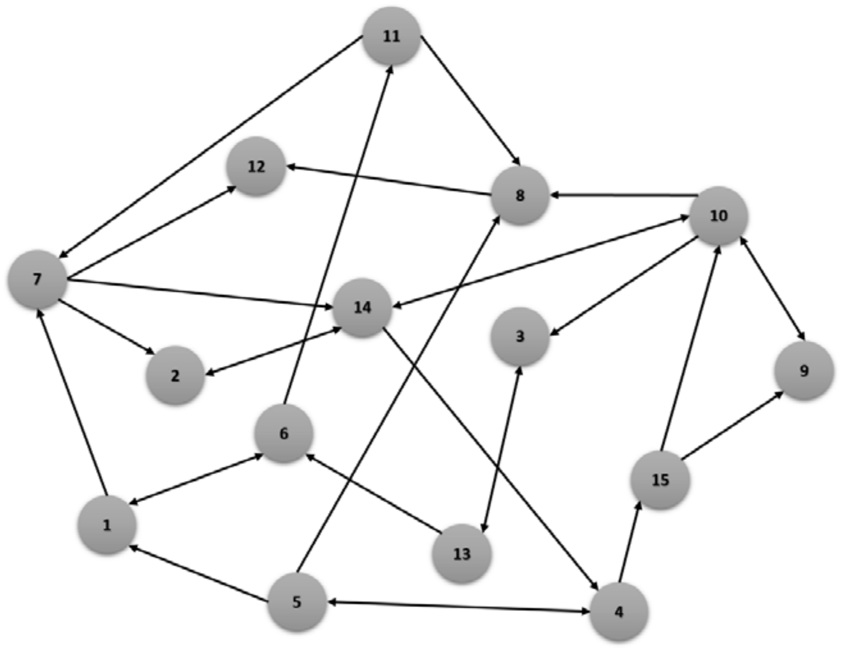
Methods & Procedure

In this section we present a process that an experienced tester could utilize while optimizing the total knowledge to be gained. This process allows the tester to perform robustness testing of a system while considering constraints such as resources, effort, and cost. For this process we assume that all possible tests that could be performed are identified and well understood.

Test Optimization Process

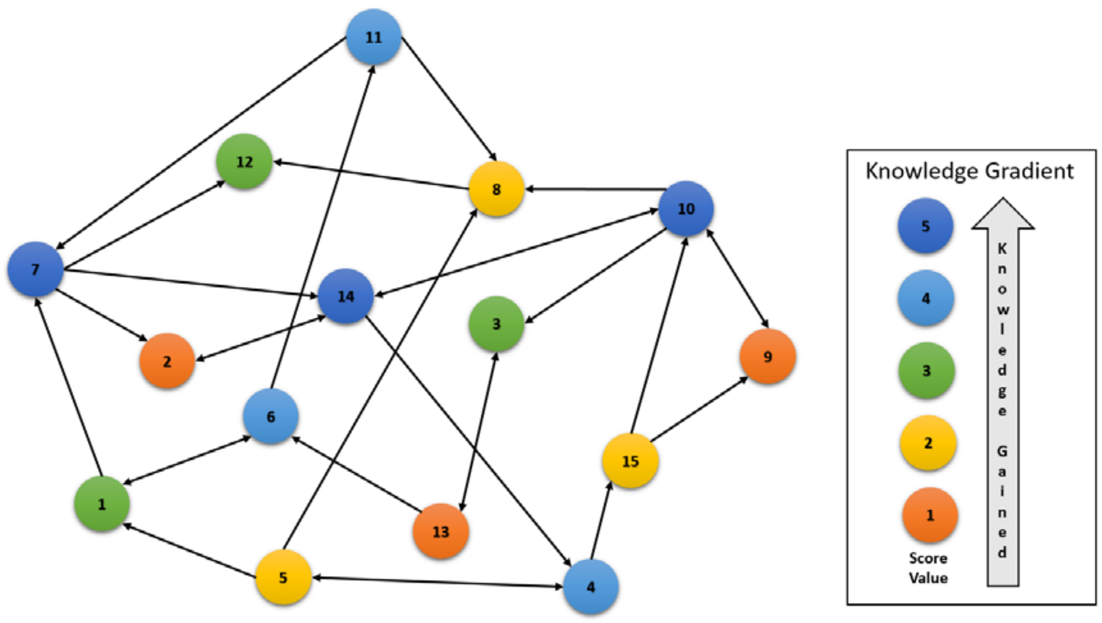
Step 1: Establishing the Test Network

Each test is assigned a unique test identification number, if not already done, a tester can do this. Then, they identify all successor tests in which the current tests outcome will influence their KG index values. This information will be used later to create the test network to be used in the algorithm. Below is a visual example of a size 15 network.



Step 2: Knowledge Gradient Index

A well-defined KG index needs to be created to represent the knowledge that a given tests tells us about the system of interest. This is extremely subjective and should be done by an expert who is very familiar with T&E, as well as the type of system being tested. For this network of tests, we will be using a generic 1 to 5 value gradient index, where 5 is the greatest amount of knowledge that can be initially gained. In order to keep a consistent system of measurement, the KG index must have a very specific KG policy developed that describes how test results are valued at every level of the index. See the example below, and note that there is no description provided for each KG index value due to no creation of a KG policy.



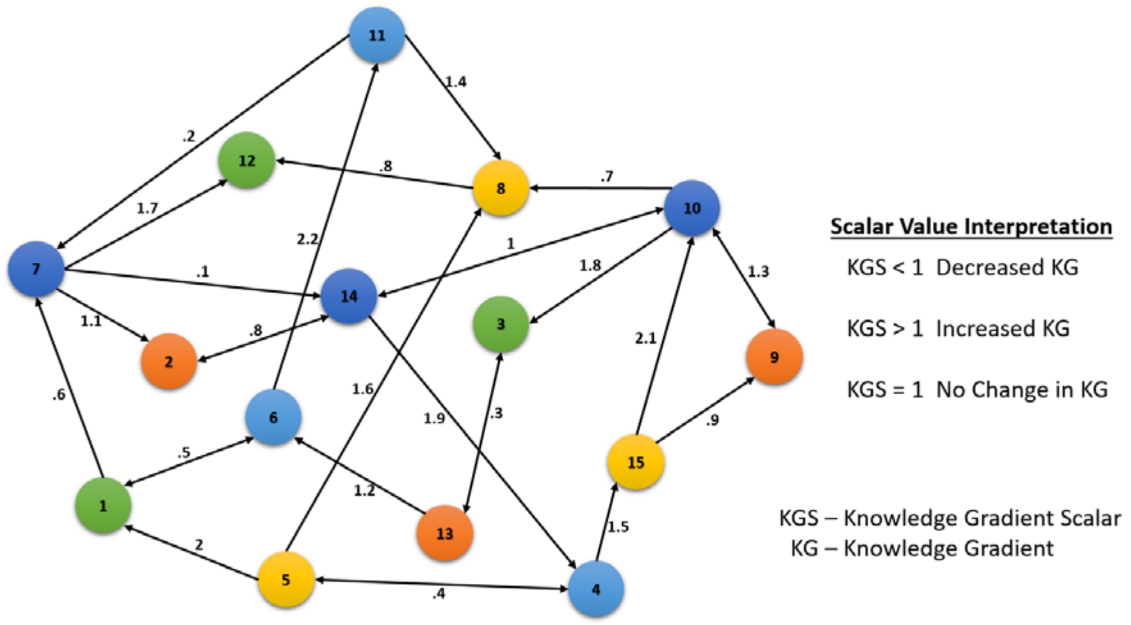
Step 3: Assigning Expected Effort and Cost

Every test should have an expected cost and the estimated effort associated with it. A test node could contain a variety of different information depending on what constraints you want to optimize to. For this project we are going to look at cost and/or effort as it affects overall knowledge of the system. Once this step is complete, all node information for the network is defined.

- $31,600.00
- 16 Person Hours
- 2 Knowledge Gradient (KG) Units
- All Other Pertinent Test Information

Step 4: Knowledge Gradient Scalars

Using the results of Step 1, for every test a tester will assign each of the identified successor tests an initial KG index scalar (KGS). If the test results for a given network are binary, pass or fail, 1 or 0, then the tester will just assign a KGS value to each of the test network arcs. However, if the tests results are complex or have multiple criteria to be considered passing than a KG policy is used create a binary result. In this case the KGS becomes a variable, we will avoid this situation for this project. Overall, this is extremely subjective and needs to be well thought out by an expert tester. Below is a finished visual example of a test network.



Step 5: Inputting the Network Data

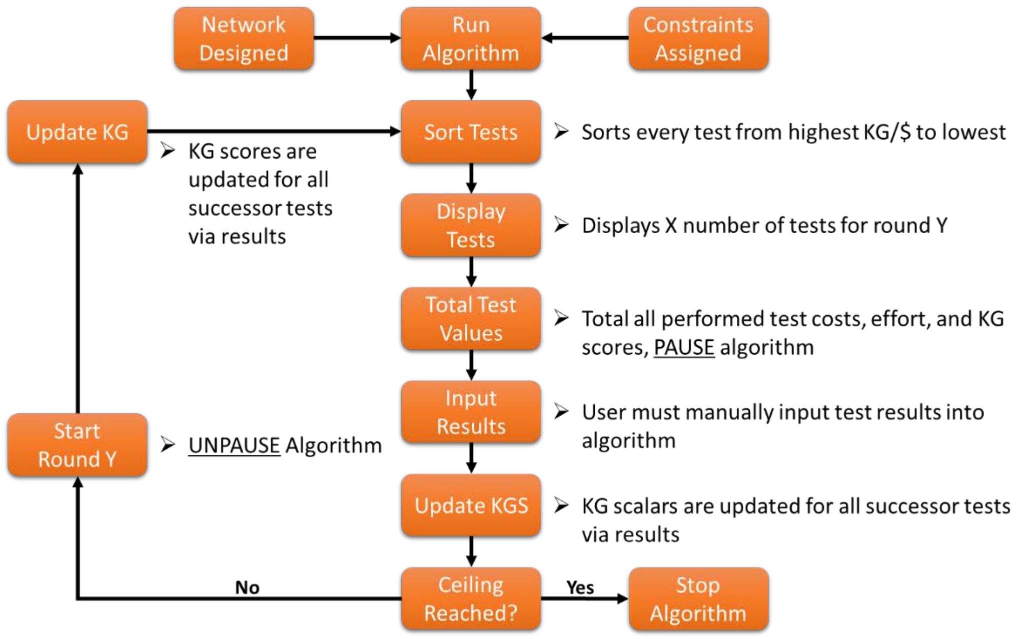
There are many different ways to input the network data into the algorithm. The simplest way is to have a friendly user interface that doesn't require the user to have to code this data directly in. For the proof of concept provided later in this paper, we use adjacency matrices for the arc data, lists for the node data, and made it easy to input by using an easily customizable user interface. Regardless of programming language or interface design, once a tester has fully completed Steps 1 thru 4, they need to input the information into the algorithm.

Step 6: Run Algorithm

**If returning from Step 9, skip the following and just run the algorithm.

Before you run the algorithm, constraints must be assigned and inputted. These constraints could include a maximum cost ceiling, a maximum effort ceiling (or time allowed), and the limitation of testing resources. This last constraint is essentially the maximum amount of tests you want to run concurrently. With these three constraints assigned in the algorithm, it is ready to run.

The following is a visual representation of the algorithm function to perform the optimization. This flow chart includes Steps 7, 8, and 9.


Network Designed
Run Algorithm
Constraints Assigned
Update KG
Sort Tests
➢ Sorts every test from highest KG/$ to lowest
➢ KG scores are updated for all successor tests via results
Display Tests
➢ Displays X number of tests for round Y
Total Test Values
➢ Total all performed test costs, effort, and KG scores, PAUSE algorithm
Start Round Y
➢ UNPAUSE Algorithm
Input Results
➢ User must manually input test results into algorithm
Update KGS
➢ KG scalars are updated for all successor tests via results
No
Ceiling Reached?
Yes
Stop Algorithm

Step 7: Perform Suggested Tests

After running the algorithm, it will return the recommend tests to perform for the current round of testing. These tests will be selected based on their KG index value per dollar. As tests are being completed, proceed to Step 8.

Step 8: Input Test Results

No matter how the user interface was created, there needs to be a results section to enter the final test results. The complexity of this input area will be dependent upon the type of test results you are returning; this is contingent on the KG policy developed. For this project a simple binary test result will be used, pass or fail. The KG policy to value the KGS's could be an additional algorithm just to interpret testing results. We leave this for a tester or student who desires to continue the development the knowledge gradient for a specific system. Input the test results.

Step 9: Repeat Steps 6 – 8

After every round of testing is complete, you will need to repeat Steps 6 thru 8. This will continue until either the cost ceiling or effort ceiling has been reached. With the proof of concept provided with this project, you can see after every round how much has been spent, how much effort has been used, and how much knowledge has been gained about the entire system. Once a ceiling has been reached and multiple rounds have been performed you can see the totals for all.

Assumptions & Limitations

There is a clear and evident limitation to this procedure and algorithm solution – it has to be customized for every type of system under robustness testing, especially the KG policy. It would be ideal to make a one-size-fits-all algorithm based solution that is simple and can accommodate the majority of robustness testing situations, but that is not feasible.

It could be possible when performing robustness testing on some types of widgets, to create a one-size-fits-all algorithm solution (like durability or reliability testing). For example, if performing durability testing on an automobile, it may be possible to make a solution that could be used on every type of automobile. This is because every automobile has the same basic overall design and would undergo the same or similar set of tests to determine its robustness (durability). The variation would be minimal.

Due to the customized nature of the algorithms implementation, the proof of concept provided with this report is simplified and will represent a generic version of this test optimization process.

Proof of Concept Simplification

The goal of this project is to create a proof of concept that validates the T&E situation of robustness testing. Simply put, having far more tests to conduct then realistically feasible under a set of constraints. Therefore, the proof of concept shown in the next section and provided with this report has been simplified to only validate the concept and algorithm. The following is a list of simplifications and/or assumptions:

- All associated test metrics (identification #, fixed cost, variable cost, required effort, and KG score) are randomly generated and assigned
    - KG index values are a 1 to 10 scale, were 10 is max knowledge
- Test relationships or network structure was randomly generated
- KGS values are static and randomly generated
- All tests are assumed to have passed the first time
- Algorithm does not pause between testing rounds because KGS's are static and predetermined, therefore no need for a manual update
- Effort and total effort are only to view, not used as a ceiling constraint

Programming Language

To make the proof of concept more user friendly, MS Excel was chosen for its convenient user interface and its developer tool, VBA. Another reason it was selected is everyone has easy access to MS Excel, so the file accompanying this project paper can be easily viewed, used, and modified by anyone. With the limitation of being unable to create a one-size-fits-all algorithm tool, MS Excel can easily be customized to any testing situation.

Proof of Concept

Step 1: Establishing the Test Network

The network being optimized is constructed of a 100 tests, and the network structure was randomly generated. The following image is a screen shot of only the first 25 tests.

A "1" indicates an arc between test nodes, a "0" indicates no arc between tests nodes

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0
5	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
13	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
18	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
19	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
21	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Steps 2 & 3: Scoring the Knowledge Gradient Index (KG) and Assigning Expected Effort and Cost

Everything in the grey boxes is randomly generated. Any of these values can be easily changed as desired. Also, this view is less cumbersome than in step 1, this chart shows the number of successors and predecessors from the test relationship network. The KG/$ is calculated automatically from the grey boxed inputs. The first 40 tests are shown below.

Test #	Test Cost with No Labor	Person Hours to Complete Test	Test Cost with Labor	Knowledge Gradient (KG) (1 to 10 scale)	# of Successor Tests	# of Predecessor Tests	KG/$ x 1000
100	$4,248,299.00	20489	$6,297,199.00	724	937	937	14.686017641
1	$24,036.00	170	$41,036.00	9	8	15	0.219319622
2	$7,813.00	55	$13,313.00	4	9	8	0.300458199
3	$55,593.00	241	$79,693.00	9	12	7	0.112933382
4	$68,505.00	357	$104,205.00	7	9	11	0.067175279
5	$70,567.00	278	$98,367.00	6	7	12	0.060996066
6	$14,769.00	85	$23,269.00	8	12	6	0.343805063
7	$52,680.00	224	$75,080.00	4	7	7	0.053276505
8	$69,140.00	321	$101,240.00	8	7	7	0.079020150
9	$55,829.00	228	$78,629.00	8	7	7	0.101743631
10	$63,104.00	325	$95,604.00	9	10	10	0.094138321
11	$78,561.00	331	$111,661.00	7	5	6	0.062689748
12	$37,052.00	76	$44,652.00	6	10	12	0.134372481
13	$63,642.00	84	$72,042.00	7	9	9	0.097165542
14	$32,323.00	337	$66,023.00	8	9	9	0.121169895
15	$25,403.00	143	$39,703.00	10	9	5	0.251870136
16	$56,058.00	70	$63,058.00	6	12	7	0.095150496
17	$51,790.00	321	$83,890.00	7	8	12	0.083442603
18	$11,635.00	109	$22,535.00	8	11	12	0.355003328
19	$49,288.00	310	$80,288.00	4	13	6	0.049820646
20	$43,708.00	94	$53,108.00	6	10	11	0.112977329
21	$35,109.00	237	$58,809.00	5	5	14	0.085021000
22	$54,898.00	288	$83,698.00	5	9	12	0.059738584
23	$53,627.00	340	$87,627.00	7	4	8	0.079884054
24	$15,068.00	123	$27,368.00	6	9	10	0.219234142
25	$34,910.00	53	$40,210.00	9	10	8	0.223824919
26	$20,699.00	133	$33,999.00	9	10	10	0.264713668
27	$54,515.00	110	$65,515.00	6	10	10	0.091582080
28	$67,469.00	171	$84,569.00	5	16	12	0.059123319
29	$48,978.00	323	$81,278.00	5	9	14	0.061517262
30	$29,417.00	219	$51,317.00	7	11	15	0.136407039
31	$24,884.00	166	$41,484.00	9	10	8	0.216951114
32	$29,437.00	83	$37,737.00	9	14	10	0.238492726
33	$34,953.00	110	$45,953.00	10	13	6	0.217613649
34	$29,703.00	288	$58,503.00	8	8	9	0.136745124
35	$67,356.00	383	$105,656.00	7	10	5	0.066252745
36	$36,830.00	229	$59,730.00	4	5	12	0.066968023
37	$49,502.00	139	$63,402.00	8	10	14	0.126178985
38	$43,876.00	186	$62,476.00	5	6	11	0.080030732
39	$8,983.00	37	$12,683.00	5	9	14	0.394228495
40	$74,470.00	342	$108,670.00	8	12	10	0.073617374

Step 4: Knowledge Gradient Scalars (KGS)

These KGS values, as mentioned previously, are completely static and do not change via pending results. All of these values were randomly generated between 0 and 2. Below is a section of the KGS's for the 100 test network created for this proof of concept.

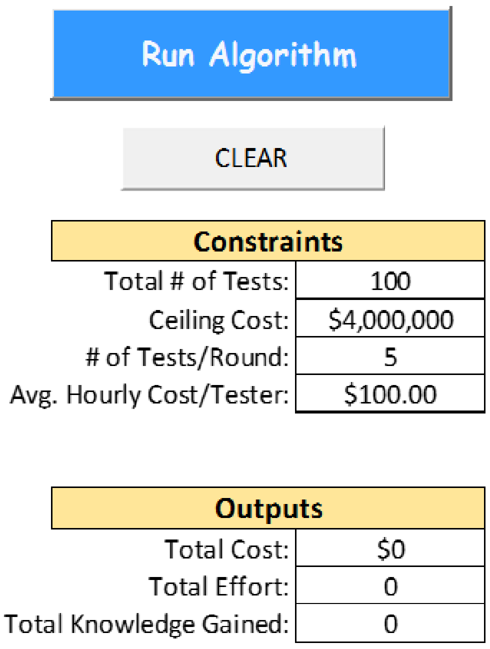
Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.00	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00	1.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.43
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00	1.77	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.59	1.43	0.00	0.00
5	0.00	0.00	0.00	0.00	1.23	0.00	0.00	0.04	0.00	0.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	1.38	0.00	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.15	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.72
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.49	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.54	0.00	0.00	0.00	0.65
13	0.00	0.00	0.70	0.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	1.05	0.00	0.00	0.00	1.41	0.00	0.82	0.00	0.00	0.00	0.12	0.00	0.00	0.00	1.27	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.07	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.76	0.00	0.00	0.00	0.00	0.00	0.00	0.00
18	0.00	1.57	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.21	0.00	1.90	0.00	0.00
19	0.00	0.00	1.76	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.27	0.00	0.00	0.66	1.78	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19	0.00	0.00	0.00	0.61	0.85	0.00	0.00	0.00	0.00	0.00	0.00
21	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
22	0.00	0.37	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
23	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.13	0.00	0.00
25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.01	0.00	1.33	0.00	0.00	0.00
26	0.46	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.44	0.00	0.00	0.00	0.00	0.00	0.00	1.82
27	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.65	0.00	0.00	0.00	0.00	0.37	0.00	0.00	0.00
28	0.00	0.00	0.00	0.00	1.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00
29	0.00	0.00	0.00	0.39	0.00	0.00	0.00	0.01	0.00	0.00	0.00	1.70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	0.75	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Step 5: Inputting the Network Data

Steps 1 thru 4 shows how to input the network data into the MS Excel interface.

Step 6: Run Algorithm

Before running the algorithm, the constraints must be inputted as shown below. The algorithm has been programmed for a 100 test network, therefore never change the "Total # of Tests:" cell, which is shown as an adjustable constraint. It is not. The constraint called "# of Tests/Round" is the testing constraint. This limits the amount of tests that can be performed at one time (a given round of tests.)



Constraints	
Total # of Tests:	100
Ceiling Cost:	$4,000,000
# of Tests/Round:	5
Avg. Hourly Cost/Tester:	$100.00

Outputs	
Total Cost:	$0
Total Effort:	0
Total Knowledge Gained:	0

Once all constraints have been entered, run the algorithm by pressing the “run Algorithm” button. Note that if tests round results are visible, the “CLEAR” button must be used before running the algorithm. This will be abundantly clear in the next step.

Step 7: Perform Suggested Tests

Below is the output displayed on the “Interface” tab contained within the proof of concept. This displays the first round of recommend tests to be performed concurrently, including associated information.

Test Order to Meet Parameters							
Test #	**Test Cost with No Labor**	**Person Hours to Complete Test**	**Test Cost with Labor**	**Knowledge Gradient (KG) (1 to 10 scale)**	**# of Successor Tests**	**# of Predecessor Tests**	**KG/$ x 1000**
Round 1:							
78	$7,562.00	50	$12,562.00	10	7	8	0.7960515841
99	$13,508.00	66	$20,108.00	8	8	10	0.3978516014
39	$8,983.00	37	$12,683.00	5	9	14	0.3942284948
84	$11,421.00	40	$15,421.00	6	10	10	0.3890798262
63	$15,099.00	61	$21,199.00	8	8	12	0.3773762913

Step 8: Input Test Results

On the previous step, step 7, the algorithm would pause if a tester was actually using this process, procedure, and algorithm to optimize the knowledge about a system. While paused the results would be inputted into the algorithm so the KGS’s for the successor tests can be manually updated. This proof of concept does not pause because all tests are assumed to pass which creates static KGS’s and simplifies the algorithm.

Step 9: Repeat Steps 6 – 8

The proof of concept repeats steps 6 thru 8 until the cost ceiling constraint has been reached. Whatever that value is, the algorithm stops once it displays the last test that exceeded the cost ceiling. Basically, if the ceiling cost is $100 dollars and the current total is $91, and assuming the next test cost $15 to perform, the algorithm would recommend performing this test, and the total spent on testing would be $106. The algorithm is set to spend all of the ceiling cost value.

Results

Using the constraints that can be seen on the image in step 6, here are the final round by round results:

Test Order to Meet Parameters							
Test #	Test Cost with No Labor	Person Hours to Complete Test	Test Cost with Labor	Knowledge Gradient (KG) (1 to 10 scale)	# of Successor Tests	# of Predecessor Tests	KG/$ x 1000
Round 1:							
78	$7,562.00	50	$12,562.00	10	7	8	0.7960515841
99	$13,508.00	66	$20,108.00	8	8	10	0.3978516014
39	$8,983.00	37	$12,683.00	5	9	14	0.3942284948
84	$11,421.00	40	$15,421.00	6	10	10	0.3890798262
63	$15,099.00	61	$21,199.00	8	8	12	0.3773762913
Round 2:							
18	$11,635.00	109	$22,535.00	13.41614818	11	12	0.5953471568
79	$29,760.00	101	$39,860.00	15.19587204	10	4	0.3812311098
80	$15,426.00	77	$23,126.00	8	9	12	0.3459309868
6	$14,769.00	85	$23,269.00	8	12	6	0.3438050625
2	$7,813.00	55	$13,313.00	4	9	8	0.3004581988
Round 3:							
91	$33,746.00	273	$61,046.00	18.46964821	10	5	0.3025529635
34	$29,703.00	288	$58,503.00	15.66614675	8	9	0.2677836479
26	$20,699.00	133	$33,999.00	9	10	10	0.2647136680
66	$24,664.00	114	$36,064.00	9	9	5	0.2495563443
32	$29,437.00	83	$37,737.00	9	14	10	0.2384927260
Round 4:							
62	$24,070.00	276	$51,670.00	18.77429532	10	12	0.3633500158
98	$36,835.00	138	$50,635.00	12.36962494	7	6	0.2442900156
47	$45,553.00	307	$76,253.00	17.9647695	12	9	0.2355942651
31	$24,884.00	166	$41,484.00	9.709146719	10	8	0.2340455771
85	$26,290.00	212	$47,490.00	11.00391975	11	8	0.2317102494
Round 5:							
90	$38,010.00	270	$65,010.00	14.79434128	10	8	0.2275702396
59	$38,383.00	361	$74,483.00	15.40307924	14	6	0.2067999307
14	$32,323.00	337	$66,023.00	11.92602309	9	9	0.1806343712
64	$41,244.00	120	$53,244.00	9.23725851	13	14	0.1734891915
41	$52,167.00	222	$74,367.00	12.66168027	15	6	0.1702593928
Round 6:							
52	$35,055.00	276	$62,655.00	10	12	7	0.1596041816
23	$53,627.00	340	$87,627.00	13.86413634	4	8	0.1582176309
88	$52,468.00	48	$57,268.00	9	11	7	0.1571558287
73	$60,088.00	264	$86,488.00	12.4897378	12	13	0.1444100662
92	$32,151.00	329	$65,051.00	9.053421813	6	12	0.1391742143
Round 7:							
72	$54,751.00	105	$65,251.00	14.84998627	15	4	0.2275825086
82	$64,048.00	108	$74,848.00	10.56398447	6	6	0.1411391683
48	$34,839.00	160	$50,839.00	7.074814671	10	5	0.1391611690
1	$24,036.00	170	$41,036.00	5.541901938	8	15	0.1350497597
29	$48,978.00	323	$81,278.00	10.43128572	9	14	0.1283408267
Round 8:							
37	$49,502.00	139	$63,402.00	12.28203952	10	14	0.1937169099
10	$63,104.00	325	$95,604.00	13.32570212	10	10	0.1393843576
30	$29,417.00	219	$51,317.00	6.682373959	11	15	0.1302175489
54	$61,522.00	161	$77,622.00	9.851865347	7	5	0.1269210449
67	$71,615.00	233	$94,915.00	11.60010203	12	8	0.1222156880
Round 9:							
60	$40,163.00	198	$59,963.00	10.73226259	12	8	0.1789814150
42	$30,122.00	342	$64,322.00	10.48296749	12	11	0.1629763921
96	$34,166.00	347	$68,866.00	7.758356921	10	13	0.1126587419
16	$56,058.00	70	$63,058.00	6.930850224	12	7	0.1099123065
11	$78,561.00	331	$111,661.00	11.56701554	5	6	0.1035904706

Round 10:							
13	$63,642.00	84	$72,042.00	8.747415279	9	9	0.1214210499
51	$59,615.00	315	$91,115.00	9.864680618	11	7	0.1082662637
33	$34,953.00	110	$45,953.00	4.687299829	13	6	0.1020020418
28	$67,469.00	171	$84,569.00	8.285576672	16	12	0.0979741592
65	$28,478.00	151	$43,578.00	4.226974454	5	5	0.0969978993
Round 11:							
83	$54,318.00	383	$92,618.00	12.9718796	5	16	0.1400578678
5	$70,567.00	278	$98,367.00	12.34231374	7	12	0.1254720968
86	$69,828.00	170	$86,828.00	7.353564473	12	8	0.0846911650
36	$36,830.00	229	$59,730.00	4.857932094	5	12	0.0813315268
94	$30,504.00	126	$43,104.00	3.410548684	13	9	0.0791237167
Round 12:							
71	$31,113.00	75	$38,613.00	3.455298042	9	15	0.0894853558
15	$25,403.00	143	$39,703.00	3.370511478	9	5	0.0848931184
93	$24,082.00	283	$52,382.00	3.873943752	4	7	0.0739556289
69	$45,428.00	63	$51,728.00	3.520224038	8	10	0.0680525835
25	$34,910.00	53	$40,210.00	2.693944143	10	8	0.0669968700
Round 13:							
17	$51,790.00	321	$83,890.00	6.032079735	8	12	0.0719046339
46	$71,216.00	291	$100,316.00	6.498207196	10	13	0.0647773754
87	$49,414.00	252	$74,614.00	3.770829196	6	10	0.0505378239
45	$55,504.00	286	$84,104.00	3.314062836	9	7	0.0394043427
35	$67,356.00	383	$105,656.00	3.898871433	10	5	0.0369015620
Round 14:							
56	$61,081.00	235	$84,581.00	5.282694862	6	13	0.0624572287
55	$40,981.00	95	$50,481.00	1.717235909	9	9	0.0340174701
21	$35,109.00	237	$58,809.00	1.97151369	5	14	0.0335240132
43	$24,406.00	334	$57,806.00	1.62055177	8	7	0.0280343177
24	$15,068.00	123	$27,368.00	0.516795608	9	10	0.0188832069
Round 15:							
53	$25,463.00	263	$51,763.00	1.313377286	11	7	0.0253728974
76	$57,288.00	382	$95,488.00	1.773499244	5	6	0.0185730065
20	$43,708.00	94	$53,108.00	0.855634772	10	11	0.0161112219
77	$43,655.00	236	$67,255.00	0.756002969	11	9	0.0112408441
38	$43,876.00	186	$62,476.00	0.526519482	6	11	0.0084275479
Round 16:							
27	$54,515.00	110	$65,515.00	0.419866017	10	10	0.0064087006
68	$55,408.00	140	$69,408.00	0.438371045	7	7	0.0063158576
40	$74,470.00	342	$108,670.00	0.619603367	12	10	0.0057016966
95	$62,843.00	82	$71,043.00	0.3534205	8	9	0.0049747406
89	$55,964.00	274	$83,364.00	0.333988549	12	11	0.0040063882
Round 17:							
75	$27,717.00	123	$40,017.00	0.128481811	10	9	0.0032106807
74	$45,372.00	256	$70,972.00	0.198662301	10	10	0.0027991645
3	$55,593.00	241	$79,693.00	0.146608362	12	7	0.0018396642

Outputs	
Total Cost:	$5,042,092
Total Effort:	16389
Total Knowledge Gained:	621

Conclusion

After testing the algorithm, there are some very interesting results on how it can not only optimize the knowledge gained about a system, but it can also significantly reduce effort and cost.

Assume a large pool of tests (100 tests) had each been assigned KG index values, and no test relationship network created, including the accompanying KGS's (meaning no KG updating). What would be the total knowledge gained about the system be if only the tests with the highest KG/$ were performed until the ceiling cost was spent? The inputs and results for this scenario are as follows:

Constraints	
Total # of Tests:	100
Ceiling Cost:	$5,000,000
# of Tests/Round:	5
Avg. Hourly Cost/Tester:	$100.00

Outputs	
Total Cost:	$5,052,173
Total Effort:	16787
Total Knowledge Gained:	655

Results with KG updating turned off

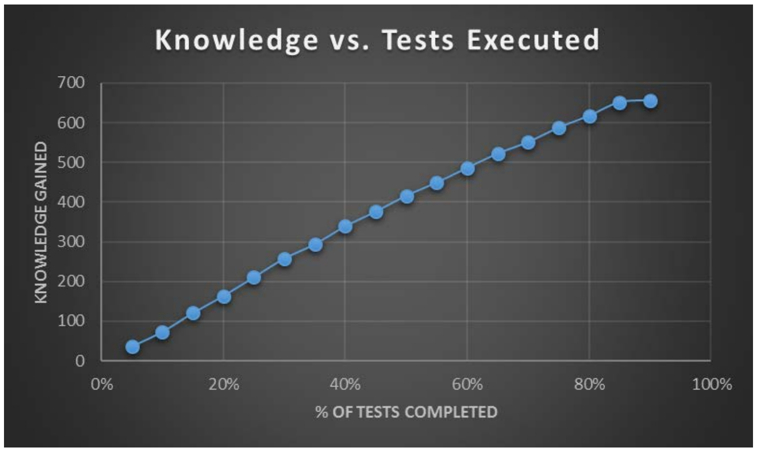
Constraints	
Total # of Tests:	100
Ceiling Cost:	$5,000,000
# of Tests/Round:	5
Avg. Hourly Cost/Tester:	$100.00

Outputs	
Total Cost:	$5,042,092
Total Effort:	16389
Total Knowledge Gained:	621

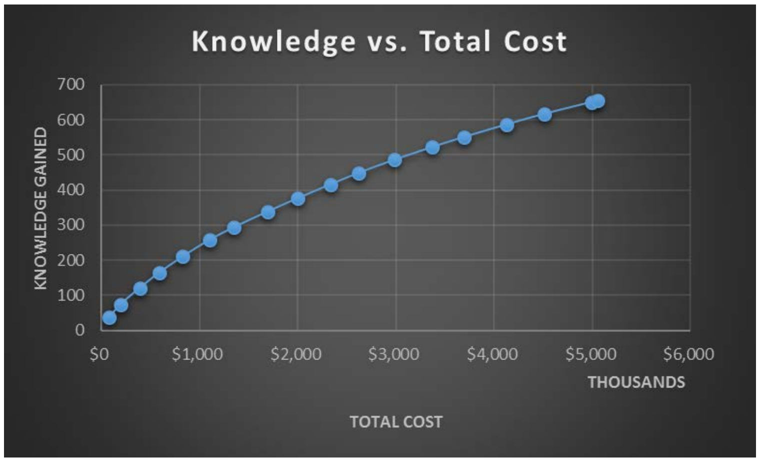
Results with KG updating turned on

Comparing the two results, you see the total knowledge gained with KG updating off and KG updating on is 655 and 621, respectively. What this proves is that a score of 655 KG isn't even attainable for this system at $5,000,000, and it proves one of Valerdi's stated limitations from earlier "*The assumption that the entire trade space is known*." This is because the first set of results isn't considering how some test results affect the KG index values of others, given a false notion of the total potential knowledge that can be gained about the system.

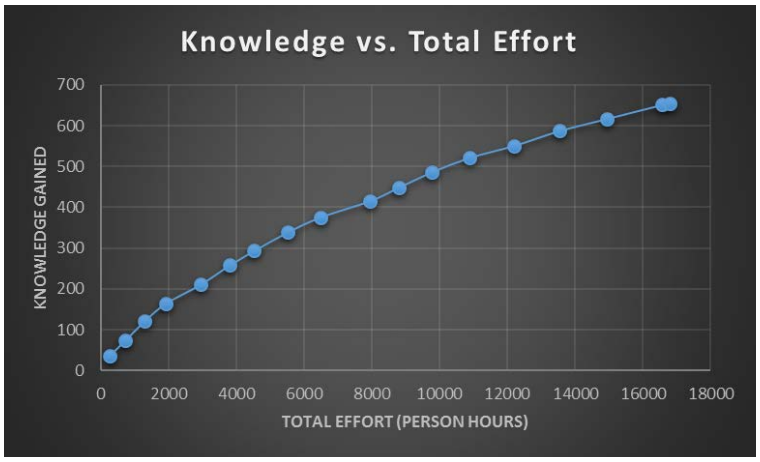
Even more interesting are the following graphs created from the above results.



Results with KG updating turned off

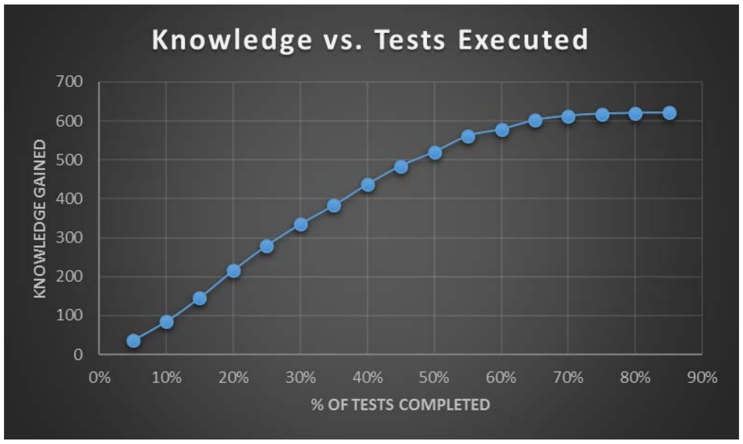


Results with KG updating turned off

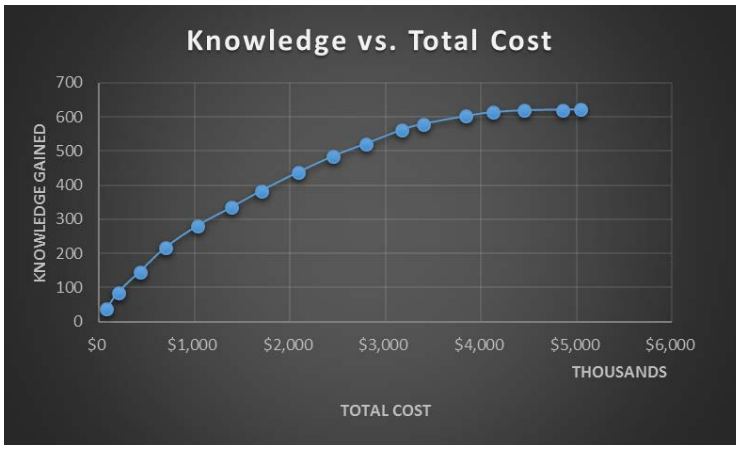


Results with KG updating turned off

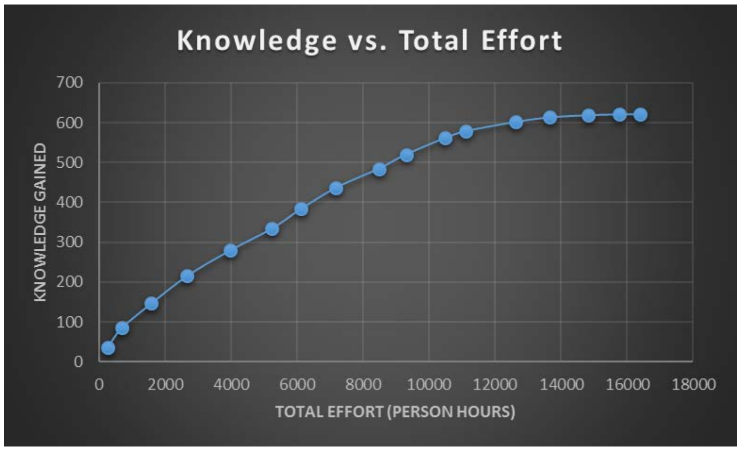
These graphical results of Knowledge vs. Tests Executed, Total Cost, and Total Effort show a nearly linear increase in total knowledge gained as you perform more and more tests. Below are the results with the KG updating turned on.



Results with KG updating turned on



Results with KG updating turned on



Results with KG updating turned on

There is a clear economy of scale displayed in these results as more and more tests are performed. In fact, observe that no matter how many tests performed beyond approximately a 600 KG value, there is no increase in the KG value. Looking at the Knowledge vs. Total Cost graph, at approximately $4,000,000 spent, there is very little knowledge to be gained about the system by spending more money. A tester being able to see this information as they proceed through T&E it can save a great deal of money and time. Let's compare the results of spending $4M vs. $5M on testing.

Constraints	
Total # of Tests:	100
Ceiling Cost:	$4,000,000
# of Tests/Round:	5
Avg. Hourly Cost/Tester:	$100.00

Constraints	
Total # of Tests:	100
Ceiling Cost:	$5,000,000
# of Tests/Round:	5
Avg. Hourly Cost/Tester:	$100.00

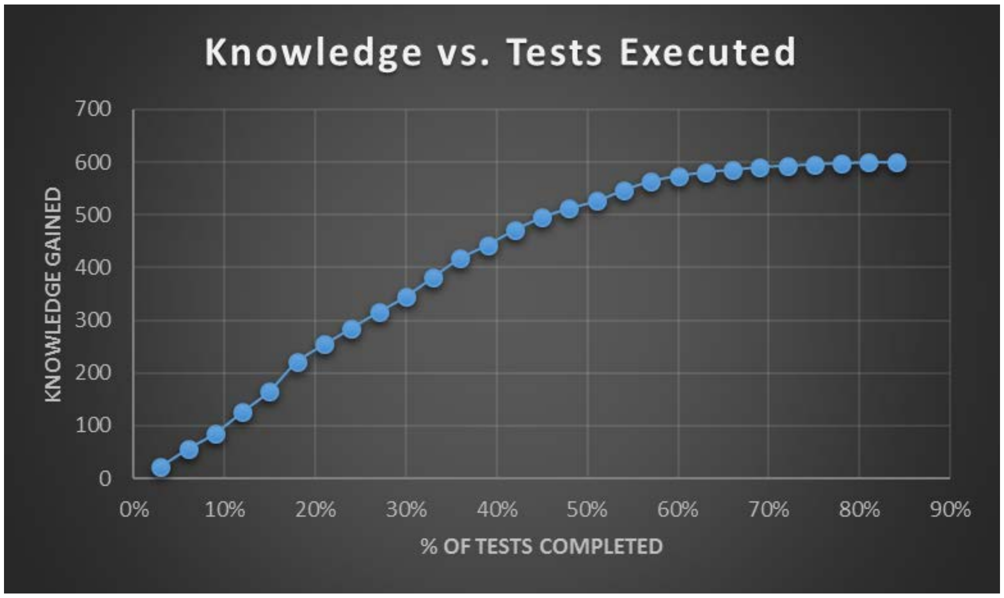
Outputs	
Total Cost:	$4,038,146
Total Effort:	13203
Total Knowledge Gained:	611

Outputs	
Total Cost:	$5,042,092
Total Effort:	16389
Total Knowledge Gained:	621

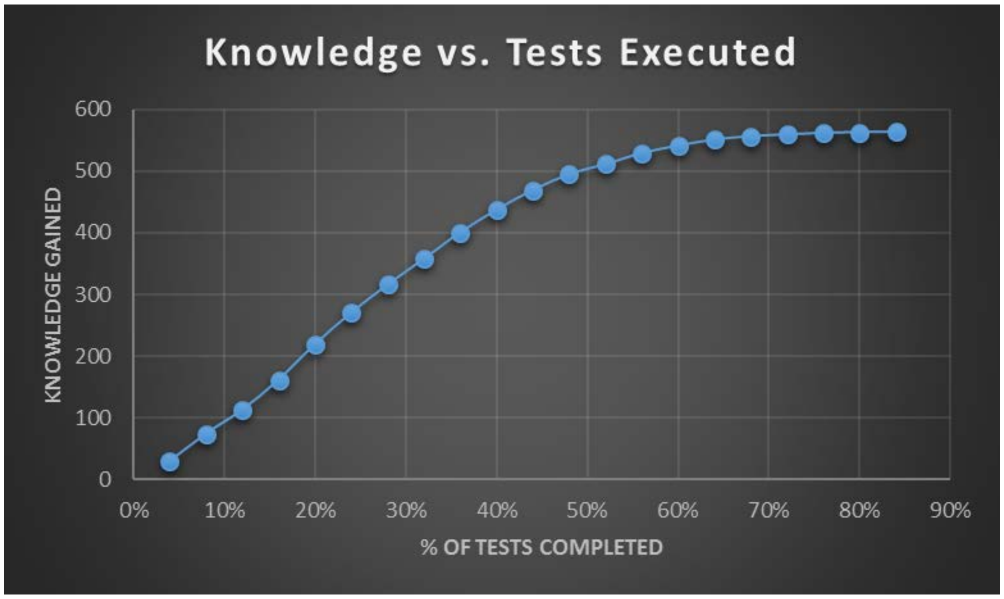
This proves that this method can save a significant amount of time and money saved while losing very little knowledge about the system undergoing robustness testing. Here are the gross savings:

	Cost Savings	Effort Savings (time)	Knowledge Lost
Difference:	$1,000,000	3186	-10
% Gained:	20%	19%	-2%

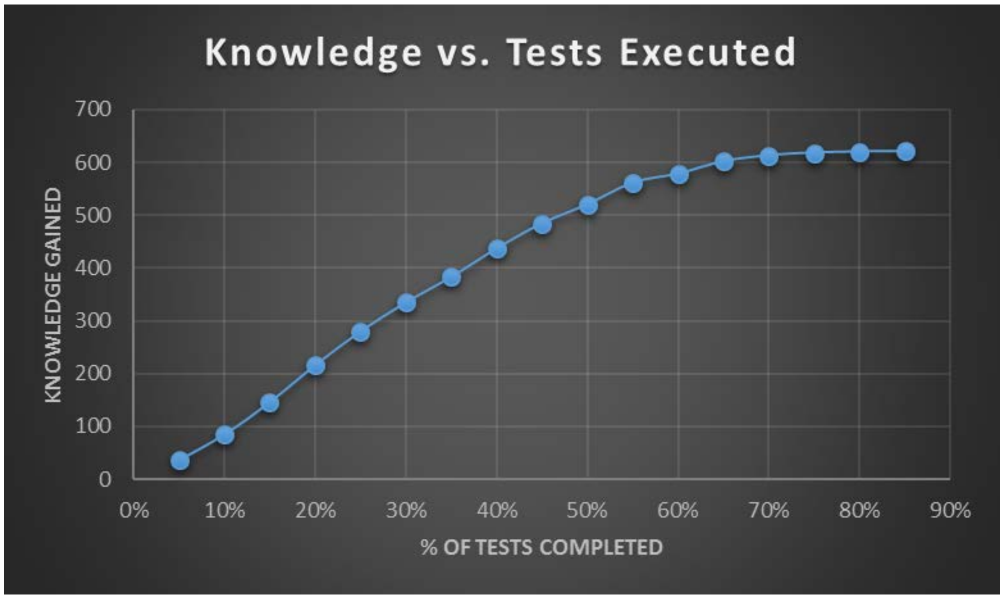
One last observation is how the total knowledge value changes depending on how many tests you perform per round. It's evident that there is an optimal number of tests to be performed at a time (tests per round), provided adequate resources are available. Holding all other constraints constant, see the following results:



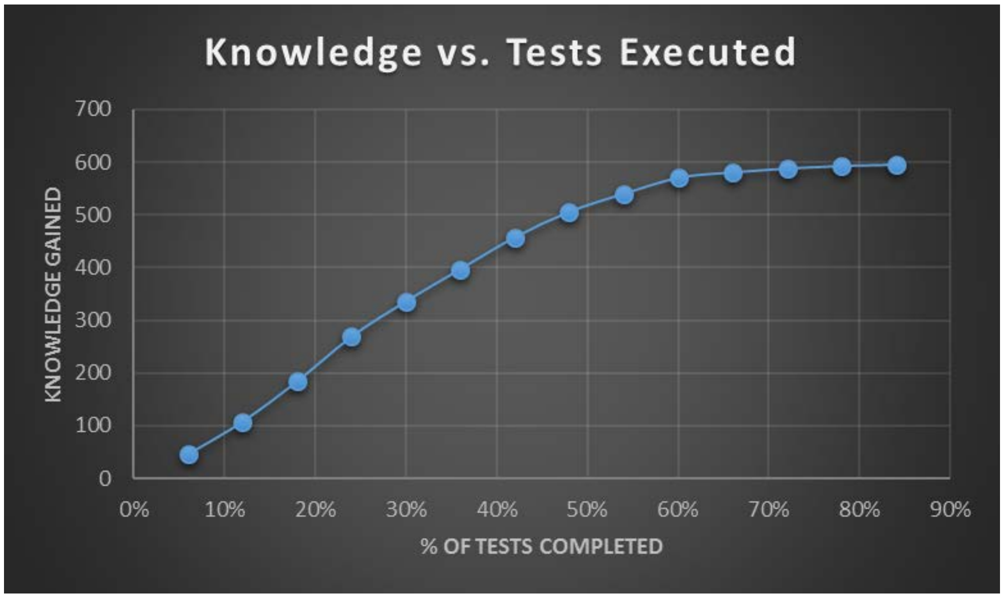
3 Tests per Round



4 Tests per Round

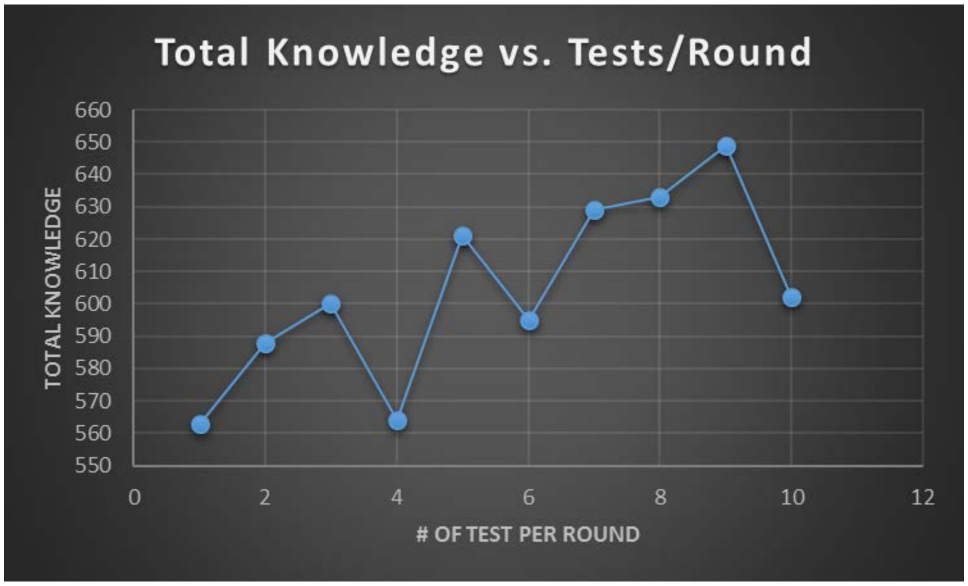


5 Tests per Round



6 Tests per Round

Here is a summary of total knowledge vs tests performed per round.



These results jump all over the place, but there is clearly an optimization to be found. One thing to note, is the total knowledge gained for 1 test per round is the maximum total knowledge value that can be obtained about the system. The reason is, the fewer tests that are ran the more granular the information is about the system. However, it would take entirely to long to perform tests one at a time, back to back. Therefore, 4 tests per round is the optimal number to run per round. The reason is because it gets closest to the true potential total knowledge of the system, while reducing the testing period.

The objective of this project was to create an algorithm that can optimize the knowledge about a complex system while performing robustness testing in the face of constraints. The results of this algorithm shows that there is a clear advantage to utilizing this T&E method of valuing testing, with obvious savings in cost and effort. Additionally, this project has revealed that there is more research and work to be done to fully develop and validate this T&E optimization approach.

Appendices

Source Code – Run Algorithm Button

```
Sheet7 - 1

Function Update(placeholder As Integer)

    Dim currentTEST As Integer, STTrow As Integer, counter As Integer, ExitKGS As Integer, KGSr
ow As Integer, KGScolumn As Integer
    Dim successorTEST As Integer, STTtestsearch As Integer

    STTrow = placeholder
    counter = 1

    While counter <= Worksheets("Interface").Cells(11, 3).Value
        currentTEST = Worksheets("Sorted Test Tracker").Cells(STTrow, 3).Value
        ExitKGS = 0
        KGSrow = 5
        While ExitKGS < 1
            KGScolumn = 5
            If Worksheets("KG Scalar").Cells(KGSrow, 4).Value = currentTEST Then
                While KGScolumn < 105
                    If Worksheets("KG Scalar").Cells(KGSrow, KGScolumn).Value > 0 Then
                        successorTEST = KGScolumn - 4
                        STTtestsearch = placeholder + Worksheets("Interface").Cells(11, 3).Valu
e

                        While STTtestsearch < 106
                            If (Worksheets("Sorted Test Tracker").Cells(STTtestsearch, 3).Value
 = successorTEST) Then
                                 Worksheets("Sorted Test Tracker").Cells(STTtestsearch, 7) = ((
Worksheets("KG Scalar").Cells(KGSrow, KGScolumn).Value) * (Worksheets("Sorted Test Tracker").Ce
lls(STTtestsearch, 7).Value))
                            End If
                        STTtestsearch = STTtestsearch + 1
                        Wend
                    End If
                KGScolumn = KGScolumn + 1
                Wend
            ExitKGS = 1
            End If

            KGSrow = KGSrow + 1
        Wend

        STTrow = STTrow + 1
        counter = counter + 1
    Wend



End Function
Function round(sortedrow As Integer, placeholder As Integer, roundKGsum As Double, summoney As
Double, sumeffort As Integer)

    Dim testsperround As Integer, rownumber As Integer, exithelp As Double

    rownumber = placeholder
    'testsperround = Worksheets("Interface").Cells(11, 3).Value + sortedrow
    counter = 1

        While counter <= Worksheets("Interface").Cells(11, 3).Value
            sortedrow = sortedrow + 1
            Worksheets("Interface").Cells(sortedrow, 5) = Worksheets("Test Metrics Sorted").Cel
ls(rownumber, 3).Value
            Worksheets("Interface").Cells(sortedrow, 6) = Worksheets("Test Metrics Sorted").Cel
ls(rownumber, 4).Value
            Worksheets("Interface").Cells(sortedrow, 7) = Worksheets("Test Metrics Sorted").Cel
ls(rownumber, 5).Value
            Worksheets("Interface").Cells(sortedrow, 8) = Worksheets("Test Metrics Sorted").Cel
ls(rownumber, 6).Value
            Worksheets("Interface").Cells(sortedrow, 9) = Worksheets("Test Metrics Sorted").Cel
ls(rownumber, 7).Value
            Worksheets("Interface").Cells(sortedrow, 10) = Worksheets("Test Metrics Sorted").Ce
lls(rownumber, 8).Value
```

```
            Worksheets("Interface").Cells(sortedrow, 11) = Worksheets("Test Metrics Sorted").Ce
lls(rownumber, 9).Value
            Worksheets("Interface").Cells(sortedrow, 12) = Worksheets("Test Metrics Sorted").Ce
lls(rownumber, 10).Value
            roundKGsum = roundKGsum + Worksheets("Test Metrics Sorted").Cells(rownumber, 7).Val
ue
            summoney = summoney + Worksheets("Test Metrics Sorted").Cells(rownumber, 6).Value
            sumeffort = sumeffort + Worksheets("Test Metrics Sorted").Cells(rownumber, 5).Value

            If Worksheets("Interface").Cells(16, 3).Value > Worksheets("Interface").Cells(10, 3
).Value Then 'exit condition
                counter = Worksheets("Interface").Cells(11, 3).Value + 1
            End If

            rownumber = rownumber + 1
            counter = counter + 1
        Wend


End Function

Function Sorter()

    Dim sortedrow As Integer
    Dim row As Integer
    Dim rownumber As Integer
    Dim maxkgperdollar As Double, lastvalue As Double

    sortedrow = 7
    lastvalue = 1

    While sortedrow <= 106
        maxkgperdollar = 0
        row = 7
        While row <= 106
            If Worksheets("Test Metrics").Cells(row, 10).Value > maxkgperdollar And Worksheets(
"Test Metrics").Cells(row, 10).Value < lastvalue Then
                maxkgperdollar = Worksheets("Test Metrics").Cells(row, 10).Value
                rownumber = row
            End If
            row = row + 1
        Wend
        lastvalue = maxkgperdollar
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 3) = Worksheets("Test Metrics").Cell
s(rownumber, 3).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 4) = Worksheets("Test Metrics").Cell
s(rownumber, 4).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 5) = Worksheets("Test Metrics").Cell
s(rownumber, 5).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 7) = Worksheets("Test Metrics").Cell
s(rownumber, 7).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 8) = Worksheets("Test Metrics").Cell
s(rownumber, 8).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 9) = Worksheets("Test Metrics").Cell
s(rownumber, 9).Value

        Worksheets("Sorted Test Tracker").Cells(sortedrow, 3) = Worksheets("Test Metrics").Cell
s(rownumber, 3).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 4) = Worksheets("Test Metrics").Cell
s(rownumber, 4).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 5) = Worksheets("Test Metrics").Cell
s(rownumber, 5).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 7) = Worksheets("Test Metrics").Cell
s(rownumber, 7).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 8) = Worksheets("Test Metrics").Cell
s(rownumber, 8).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 9) = Worksheets("Test Metrics").Cell
s(rownumber, 9).Value
```

```
        sortedrow = sortedrow + 1
    Wend


End Function

Function Sorter2(placeholder As Integer)

    Dim row As Integer, sortedrow As Integer
    Dim rownumber As Integer
    Dim maxkgperdollar As Double, lastvalue As Double

    sortedrow = placeholder
    lastvalue = 1

    While sortedrow <= 106
        maxkgperdollar = 0
        row = placeholder
        While row <= 106
            If Worksheets("Sorted Test Tracker").Cells(row, 10).Value > maxkgperdollar And Work
sheets("Sorted Test Tracker").Cells(row, 10).Value < lastvalue Then
                maxkgperdollar = Worksheets("Sorted Test Tracker").Cells(row, 10).Value
                rownumber = row
            End If
            row = row + 1
        Wend
        lastvalue = maxkgperdollar
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 3) = Worksheets("Sorted Test Tracker
").Cells(rownumber, 3).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 4) = Worksheets("Sorted Test Tracker
").Cells(rownumber, 4).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 5) = Worksheets("Sorted Test Tracker
").Cells(rownumber, 5).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 7) = Worksheets("Sorted Test Tracker
").Cells(rownumber, 7).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 8) = Worksheets("Sorted Test Tracker
").Cells(rownumber, 8).Value
        Worksheets("Test Metrics Sorted").Cells(sortedrow, 9) = Worksheets("Sorted Test Tracker
").Cells(rownumber, 9).Value
        sortedrow = sortedrow + 1
    Wend

    sortedrow = placeholder
    While sortedrow <= 106
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 3) = Worksheets("Test Metrics Sorted
").Cells(sortedrow, 3).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 4) = Worksheets("Test Metrics Sorted
").Cells(sortedrow, 4).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 5) = Worksheets("Test Metrics Sorted
").Cells(sortedrow, 5).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 7) = Worksheets("Test Metrics Sorted
").Cells(sortedrow, 7).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 8) = Worksheets("Test Metrics Sorted
").Cells(sortedrow, 8).Value
        Worksheets("Sorted Test Tracker").Cells(sortedrow, 9) = Worksheets("Test Metrics Sorted
").Cells(sortedrow, 9).Value
        sortedrow = sortedrow + 1
    Wend


End Function


Private Sub CommandButton1_Click()

    Dim roundnumber As Integer, sortedrow As Integer, placeholder As Integer, rownumber As Inte
ger, counter As Integer
    Dim exithelp As Integer, testpercent As Double, tot As Double, roundKGsum As Double, summon
```

```
ey As Double, sumeffort As Integer


    placeholder = 7
    rownumber = 7
    sortedrow = 6
    roundnumber = 1
    counter = 23
    testpercent = 0
    roundKGsum = 0
    summoney = 0
    sumeffort = 0

    Call Sorter

    While Worksheets("Interface").Cells(16, 3).Value < Worksheets("Interface").Cells(10, 3).Val
ue 'Total Acummulated Cost < Ceiling Cost
        sortedrow = sortedrow + 1 'this keeps moving down and displaying the test and info on t
he Interface
        Worksheets("Interface").Cells(sortedrow, 5).Value = "Round " & roundnumber & ":"
        roundnumber = roundnumber + 1

        Call round(sortedrow, placeholder, roundKGsum, summoney, sumeffort)

        testpercent = testpercent + ((Worksheets("Interface").Cells(11, 3).Value) / (Worksheets
("Interface").Cells(9, 3).Value))
        Worksheets("Interface").Cells(counter, 4).Value = summoney
        Worksheets("Interface").Cells(counter, 2).Value = testpercent
        Worksheets("Interface").Cells(counter, 3).Value = roundKGsum
        Worksheets("Interface").Cells(counter, 1).Value = sumeffort

        counter = counter + 1

        Call Update(placeholder) 'update from test metrics sorted

        placeholder = placeholder + Worksheets("Interface").Cells(11, 3).Value  'marks where we
 are thru the sorted tests

        'pause algorithm for manual update

        Call Sorter2(placeholder)

    Wend


End Sub
```

Source Code – Clear Button

```
Module1 - 1

Sub clear()

Dim moverow As Integer

moverow = 7

    While moverow < 300
        Worksheets("Interface").Cells(moverow, 5).ClearContents
        Worksheets("Interface").Cells(moverow, 6).ClearContents
        Worksheets("Interface").Cells(moverow, 7).ClearContents
        Worksheets("Interface").Cells(moverow, 8).ClearContents
        Worksheets("Interface").Cells(moverow, 9).ClearContents
        Worksheets("Interface").Cells(moverow, 10).ClearContents
        Worksheets("Interface").Cells(moverow, 11).ClearContents
        Worksheets("Interface").Cells(moverow, 12).ClearContents
        Worksheets("Interface").Cells(moverow + 16, 4).ClearContents
        Worksheets("Interface").Cells(moverow + 16, 2).ClearContents
        Worksheets("Interface").Cells(moverow + 16, 3).ClearContents
        Worksheets("Interface").Cells(moverow + 16, 1).ClearContents
        moverow = moverow + 1
    Wend

End Sub
```

References

[1] "V-Model," *Wikipedia, the free encyclopedia*. 15-Apr-2016.

[2] A. Shahrokni and R. Feldt, "A systematic review of software robustness," *Inf. Softw. Technol.*, vol. 55, no. 1, pp. 1–17, Jan. 2013.

[3] "TE FINAL VERSION - rev2.PDF - TECapabilities_Dec2000.pdf." .

[4] Warren B. Powell and Ilya O. Ryzhov, "Ranking and Selection," in *Optimal Learning*, First Edition., John Wiley & Sons INC, 2012.

[5] "FrazierDissertation_embeddedfonts - FrazierDissertation.pdf." .

[6] Ricardo Valerdi, "Making Big Data, Safe Data: A Test Optimization Approach." 2013.

[7] Amjad Chatila, "An Intorduction to Qualitative Resilience," University of Arizona, Systems & Industrial Engineering, May 2014.

[8] "network definition - Google Search." [Online]. Available: https://www.google.com/search?q=network+definition&ie=utf-8&oe=utf-8. [Accessed: 15-May-2016].

[9] "Social network analysis with NetworkX," *Domino Blog*, 14-Jul-2015. [Online]. Available: http://blog.dominodatalab.com/social-network-analysis-with-networkx/. [Accessed: 07-May-2016].

THIS PAGE LEFT INTENTIONALLY BLANK